TP3 : Structures alternatives et itératives

1. **Instructions conditionnées** (*if then else*)
   En *Scilab*, la syntaxe utilisée pour effectuer des instructions conditionnées par un test (alternative) est la suivante :

```
if test then
   instructions
else
   instructions
end
```

   test a une valeur logique (T ou F), par exemple le test `8>2` est vrai mais le test `x^2==-1` est faux dès lors que `x` est réel.
   Pour *Scilab* tout nombre réel non nul est vrai et 0 est faux.
   Si *test* est vrai alors les premières instructions sont réalisées sinon ce sont les secondes.
   Exemple (à tester et comprendre) :

```
birth=input("Année de naissance (ex:1996): ");
year=getdate();  //getdate() renvoie un talbeau avec
date
age=year(1)-birth;
if age >= 18 then
   disp(" ans, vous êtes majeur !", age, "Vous avez ")
else
   disp(" ans, vous êtes mineur !", age, "Vous avez ")
end
```

   Remarques :

   (a) Le `else` n'est pas obligatoire, par exemple si l'on souhaite exécuter des instructions uniquement si le *test* est vrai et ne rien faire sinon.

   (b) Il est tout à fait possible d'imbriquer plusieurs alternatives les unes à l'intérieur des autres.

2. Structures itératives *(les boucles)*

(a) boucle *for*

La syntaxe suivante :

```
for i= a:h:b
   instructions
end
```

permet de répéter les ***instructions*** lorsque `i` prend successivement les valeurs des coordonnées du vecteur ***a:h:b***.

Exemple (table de multiplication) :

```
tab=input("Table de multiplication à afficher : ");
for i=1:9
   disp(i*tab)
end
```

Exemple (somme d'inverses de nombres impairs : $1+\frac{1}{3}+\frac{1}{5}+\cdots+\frac{1}{2n+1}$*)* :

```
n=input("Entrer un nombre entier : ");
s=0;
for i=1:2:2*n+1
   s=s+1/i;  // la variable 's' accumule les termes
end
disp(s)
```

Remarques :

i. On utilise principalement les boucles *for* lorsque l'on connaît a priori le nombre d'itérations.

ii. Tout comme les instructions conditionnées, on peut imbriquer plusieurs boucle *for* les unes à l'intérieur des autres.

Exemple (Affichage des tables de 1 à 10) :

```
for i=1:10  // table de i
   for j=1:10
      disp(i*j)
   end
end
```

Remarque :
Dans ce dernier exemple, il aurait été plus commode et lisible d'écrire :

```
for k=1:10
   disp(k*(1:10))
end
```

(b) **boucle *while***

```
while test
   instructions
end
```

La syntaxe précédente permet de répéter les ***instructions*** tant que le ***test*** est vrai.
Exemple (à tester et comprendre) :

```
phrase=input("Bonjour : ", "string");
while phrase <> "au revoir"
   phrase=input("Bonjour : ", "string");
end
disp("Bye Bye !")
```

Remarques :

i. La condition peut être complexe du type : `while (x>=10)|(x<0)`
ii. L'intérêt de la boucle itérative `while` est que l'on ne sait pas toutjours, a priori, quand le ***test*** ne sera plus vrai et c'est dans ces cas-là, le plus souvent, que nous l'utiliserons.
iii. Là encore il est tout à fait possible d'imbriquer plusieurs boucles `while` les unes à l'intérieur des autres, tout comme on peut imbriquer des boucles `while` à l'intérieur de boucle `for` à l'intérieur d'instructions conditionnées...

Exemple (à tester) :

```
x=input("Entrer un nombre positif : ");
//inférieur à 16 sinon plus de 5 000 000 d'itérations
s=0;
i=1;
while s<x
   s=s+1/i;
   i=i+1;
end
disp(s,i)
```

Ce dernier exemple calcul la somme $s = 1+\frac{1}{2}+\frac{1}{3}+\dots$ tant qu'elle est inférieure à x. Dès que s dépasse x, le programme s'arrête et affiche la somme s ainsi que le nombre d'itérations i qui ont été nécessaire.

Attention : on limite $x < 16$ car si *Scilab* était suffisamment précis (ce qui n'est pas le cas), il faudrait attendre (avec un processeur 3GHz) environ 10 minutes avec $x = 20$, une centaine de jours avec $x = 30$ et plusieurs millénaires avec $x = 40$, pour obtenir un résultat.

3. Exercices

Exercice 1. On souhaite créer un programme qui demande à l'utilisateur un nombre réel x et qui affiche ensuite la racine carrée de x si $x \geqslant 0$ et un message d'alerte sinon.

Exercice 2. Créer un programme qui demande à l'utilisateur deux nombres a et b et qui résout ensuite l'équation $ax + b = 0$ (On traitera aussi le cas $a = 0$).

Exercice 3. *(Plus difficile)*
Créer un programme qui demande à l'utilisateur trois nombres réels a, b et c et qui résout ensuite l'équation $ax^2 + bx + c = 0$ (On traitera aussi le cas $a = 0$ et le cas $\Delta < 0$).

Exercice 4. En utilisant une boucle `for`, créer un programme qui demande à l'utilisateur un entier naturel n et qui affiche en retour la valeur de $n!$. Faire de même avec une boucle `while`.

Exercice 5. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 0$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = \cos(u_n)$.
Une étude (que l'on admettra ici) montre que $(u_n)_{n\in\mathbb{N}}$ converge vers une limite l qui est l'unique solution, sur $[0, \frac{\pi}{2}]$, de l'équation $x = \cos(x)$. De plus, pour tout $n \in \mathbb{N}$, l est compris entre u_n et u_{n+1}.
Écrire un programme qui affiche les valeurs successives de n et u_n et qui s'arrête lorsque u_n est une valeur approchée à 10^{-4} près de l.
Indication : On calculera u_n tant que $|u_{n+1} - u_n| > 10^{-4}$.

Exercice 6. *(Le juste prix)*
On souhaite créer le jeu suivant :
Le jeu choisit au hasard et selon une loi uniforme un nombre entre 1 et 1000. L'utilisateur tente ensuite de deviner ce nombre en moins de 10 essais. Après chaque essai le programme doit afficher une indication : plus ou moins. Le joueur gagne uniquement s'il trouve le nombre caché en moins de 10 essais. Le jeu s'arrête dès que le joueur gagne ou dès que le joueur dépasse les 10 essais.

Exercice 7. Écrire un programme qui demande à l'utilisateur un entier naturel n et qui affiche ensuite les n premières valeurs de la suite $(u_n)_n$ dans les cas suivants :

a) $u_1 = 1,\ u_2 = 1$ et pour tout $n \in \mathbb{N}^*$, $u_{n+2} = u_{n+1} + u_n$. (*Fibonacci*)

b) $u_1 = 0,\ u_2 = 1$ et pour tout $n \in \mathbb{N}^*$, $u_{n+2} = (n+1)u_{n+1} - 2u_n$.

Modifier ces programmes pour qu'ils affichent aussi la valeur de la somme des n premiers termes : $\sum_{k=1}^{n} u_k$.